下载APP

07 | 性能场景的数据到底应该做成什么样子?

2021-04-05 高楼

高楼的性能工程实战课 进入课程 >

讲述：高楼
时长 18:37 大小 17.05M

你好，我是高楼。

在性能项目中，性能数据是非常重要的输入资源。但是我经常看到有人拿着少得可怜的数据，来做比较大的压力，这显然不符合真实的场景，虽然拿到的结果很好看，但并不会得到什么有价值的结果。所以，今天我们就来讲一下性能场景中的数据到底应该做成什么样子。

在 RESAR 性能工程中，场景里使用的数据需要满足两个方面：

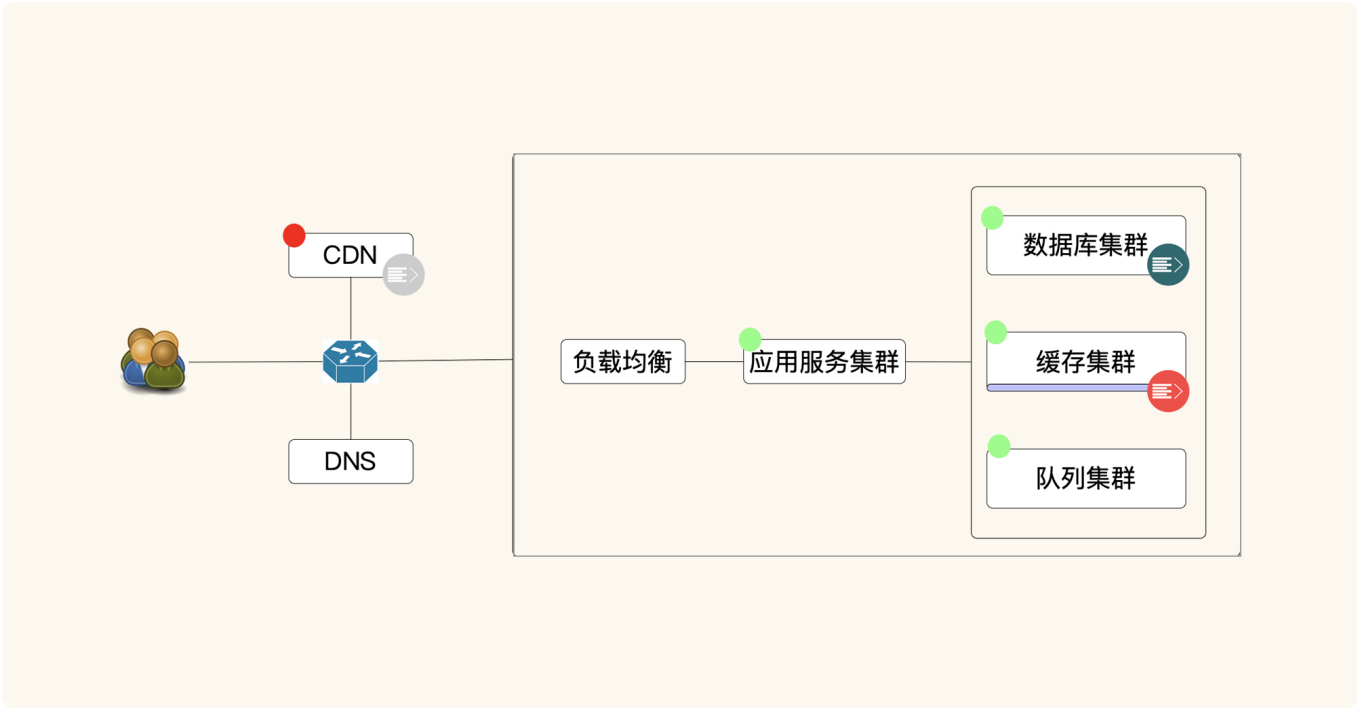
第一，数据要符合真实环境中的数据分布，因为只有这样，我们才能模拟出相应的 IO 操作；

第二，要符合真实用户输入的数据，以真正模拟出真实环境中的用户操作。

而这两个方面分别对应着两类数据：铺底数据和参数化数据。我们先来看铺底数据。

铺底数据

在通常的线上系统架构中，系统中用到的数据分为两部分：静态数据（图中红色点）和动态数据（图中绿色点），这也是我们在性能场景中需要存入的铺底数据。



从这个简单的结构图中不难看出，如果没有铺底数据，那就相当于是一个空系统。但是在生产环境中，这个系统肯定不会是空的，所以要有足够的数据在里面。如果数据不真实，我们就无法模拟出生产上有真实数据的场景，比如应用的内存占用、数据库 IO 能力、网络吞吐能力等。

其中，对于静态数据而言，我们最容易出现的问题是，一想到它占的网络带宽大，就觉得要用 CDN 了；或者是觉得不模拟静态数据，就是不符合真实场景，不支持我们的优化结果了。其实，数据放在哪里，怎么做最合理，怎么做成本最低，这些都需要综合考虑，并不是一味跟风，别人怎么做我们就要怎么做。

我曾经看到有的官方门户网站明明没几个流量，却在做技术规划的时候，非要把零星的几个图片放到 CDN 上去，以显示自己设计的架构有多先进。

我也经常看到一些企业认为网站上的图片很重要，出于不懂技术又要寻找安全感的逻辑，非要把图片都放到自己的服务器里。本来图片就很大，一张有 3～4M，用户一访问，自然就会吵吵着慢。

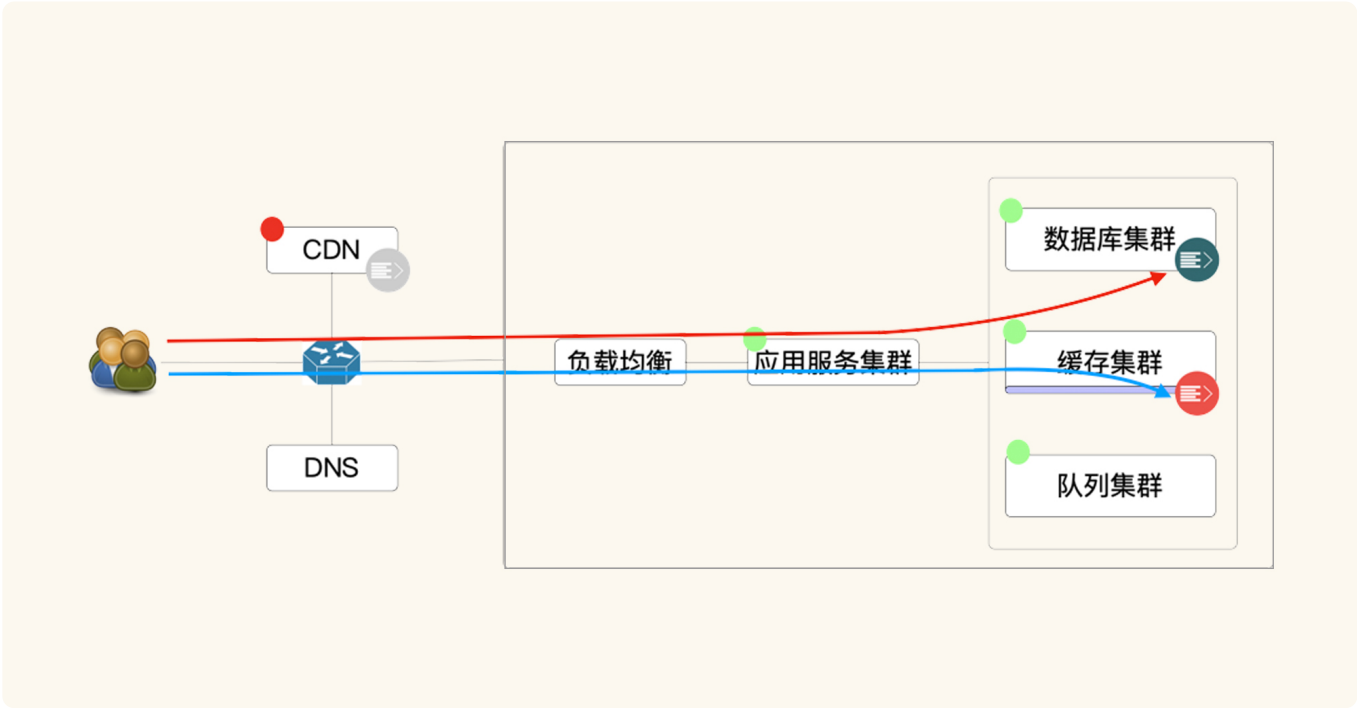
像这两种极端都不可取。要知道，当外行指使内行干活的时候，基本上没什么好结果，因为有些外行觉得只要压力发起就可以了，在细节上根本不在乎结果会怎么样。在我看来，处理这样问题的最合理的方式是先分析业务逻辑，再判断技术架构怎么实现。

我们知道，静态数据通常有两个可以存放的地方，一个是服务端的 Web 层，另一个是 CDN。对于大系统而言，流量大，网络带宽自然就要求得多。在这种情况下，数据必然要放 CDN，你也没有其他选择（当然了，你可以选择不同的 CDN 厂商）。

对于一些小的业务系统，由于用的人并不多，整体网络流量要求也少，那我们就可以把静态数据直接放到负载均衡服务器（比如 Nginx）或应用服务器中去。用户访问一次之后，后续的访问直接走本地缓存就可以了，对系统的压力也不会产生多大的影响。

静态数据讲完了，我们再来看动态数据，这就需要我们好好分析一下了，因为有些动态数据是可以放到 CDN 中的。

还是前面那张图，在我们不用任何预热加载的情况下，这些动态数据都是存放在数据库中的。当我们使用预热加载时，这些数据就会转到缓存中去（当然，这也取决于架构设计和代码实现），变成下图中这样：



所以按照这样的逻辑，真实场景中业务操作的数据量实际有多少，我们就要模拟出多少，不然会出现一些问题。当模拟数据量与实际数据量差别较大时，会对数据库、缓存等造成不同的影响。下面我列出了五点，为你具体分析一下。

对数据库压力的区别

假设线上系统中有 100 万的用户量，而我们在做压力测试时，由于没有生产数据，造数据又比较麻烦，所以就直接使用 1000 条甚至更低的用户量来做性能场景。那一个表里有 100 万条数据和 1000 条数据的差别是什么呢？我们来实际操作一下。

在这里，有一个前提条件：同样的硬件环境，同样的数据库，同样的表结构，同样的索引，只是两张表的数据不同。

两条 SQL 如下：

复制代码

```
1 select * from ob_tuning.temp1_1000 where id = '3959805';
2 select * from ob_tuning.temp2_100w where id = '3959805';
```

因为表内的数据量不同，所以结果如下：

19	13:25:36	select * from ob_tuning.temp1_1000 where id = '3959805' LIMIT 0, 2000	1 row(s) returned	0.019 sec / 0.000011...
20	13:25:36	select * from ob_tuning.temp2_100w where id = '3959805' LIMIT 0, 2000	1 row(s) returned	0.732 sec / 0.000019...

可以看到，查询时间一个是 19ms，一个是 732ms。我们不妨再来看一下表的操作细节。

第一个表（用户量为 1000）的操作细节：

第二个表（用户量为 100 万）的操作细节：

这里我们只需要对比“executing”这一行就能看到明显的差距，它告诉我们当执行这个语句时，需要的 CPU 时间明显因为数据量的增加而增加了。所以我们不难发现，如果你没有

足够的铺底数据放在性能场景中，那一开始便注定了悲剧的结果。

缓存的区别

数据量的多少在缓存中有很明显的区别，如下图所示：

也就是说，场景中用的数据量越多，缓存必然要求越大。

压力工具使用的区别

压力工具中使用的数据多少，不仅影响着压力工具本身需要的内存，同时也影响着性能场景的执行结果。这一点，我们会在后面的课程中详细讲到。

网络的区别

其实不同的数据量，不管是走缓存，还是数据库，对客户端和服务器之间的网络消耗都是差不多的。只要不是缓存在客户那边，都是要走到服务器里转一圈的。所以我们认为，数据量是多还是少，对客户端和服务器之间的网络的压力没有什么区别。如果你用的是 CDN，那可以做另外的考虑。

应用的区别

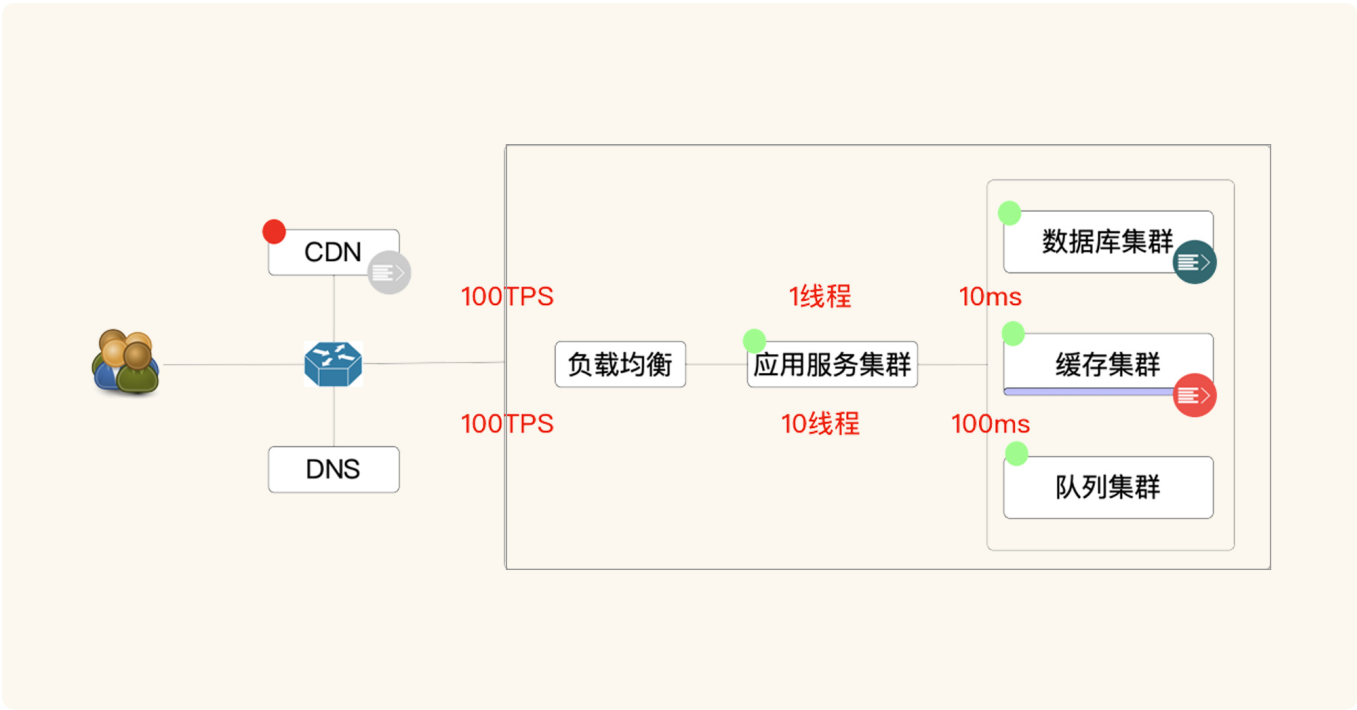
如果不是在应用中直接缓存数据，我们也认为对应用没什么区别，反正不管是什么样的请求过来，都是要到缓存或数据库中去取数据的，应用的 Self Time 不会有什么差别，方法

依旧要执行。但是，如果你的应用是直接在应用的缓存中存数据的，那就有区别了，同样也是数据量越大，对内存的要求就越大。

基于以上几点，**我们可以看到有两个比较重要的环节：数据库和缓存，这是直接的影响。**

那间接的影响有什么呢？比如说，数据在数据库中执行得慢了，在同步调用的应用中必然需要更多的应用线程来处理。

我们假设有一个 100TPS 的系统，先忽略其他时间，只看数据库时间。如果数据库执行需要 10ms，那应用只需要一个线程就能处理完了。如果数据库需要 100ms，而我们仍然想达到 100TPS，那应用就得有 10 个线程来同时处理。



与此同时，整个链路上的所有线程、队列、超时等都会因为受到数据量的影响而产生大的变化。所以，我们要想模拟出生产时候的样子，在铺底数据上一定不能含糊。

参数化数据

有了前面铺底数据的分析，我们在做参数化的时候就会明确很多。不过，在场景中应该用多少量的数据，是性能场景中最容易出问题的一个环节。

参数化数据量应该是多少，取决于场景运行多长时间。而在场景运行中，我们通常要用到两类数据：唯一性数据和可重复使用的数据。

对于唯一性数据（比如用户数据）来说，我们需要使用多少参数化数据是非常容易计算的。比如一个运行半小时的场景，TPS 如果是 100 的话，那就需要 18 万的数据量，计算过程如下：

$$数据量 = 30min \times 60s \times 100TPS = 18w$$

对于可重复使用的数据量，我们需要分析真实业务场景中是如何重复的，比如说电商系统中商品的数据量，我们在做参数化的时候就可以重复，毕竟多个人是可以同时购买同一个商品的。我们假设平均有 1000 个用户在 10 个商品中，那当我们有 18 万个用户时，就需要 1800 个商品：

$$商品数量 = 18w用户 \div 1000用户 \times 10商品 = 1800商品$$

上述就是唯一性数据量和可重复使用数据量的计算方式。

你可能会问，如果参数化数据量太大，在压力工具中处理不了怎么办？比如说我们在用 JMeter 处理文件参数化数据时，如果参数化文件太长，会导致 JMeter 消耗更多的时间。其实像这种参数化数据量要求多的情况，我们可以采用连接远程缓存（比如 Redis）或数据库（比如 MySQL）的方式来做参数化。

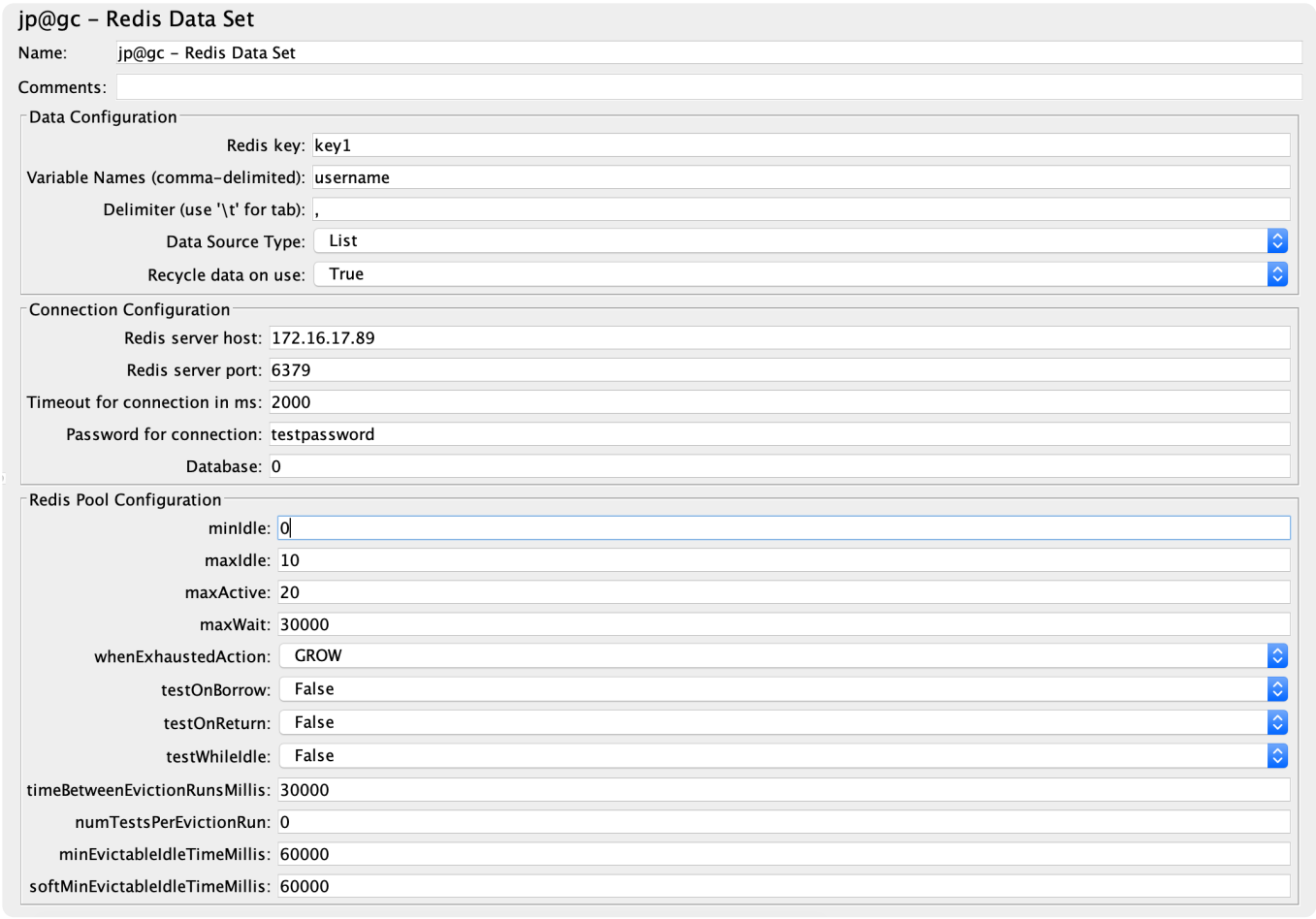
连接 Redis 做参数化

方法一：直接在 JMeter 中写 Beanshell 连接 Redis 取数据。

复制代码

```
import redis.clients.jedis.Jedis;
  //连接本地的 Redis 服务
Jedis jedis = new Jedis("172.16.106.130",30379);
log.info("服务正在运行: "+jedis.ping());
String key = vars.get("username");
String value = vars.get("token");
vars.put("tokenredis",jedis.get(key));
```

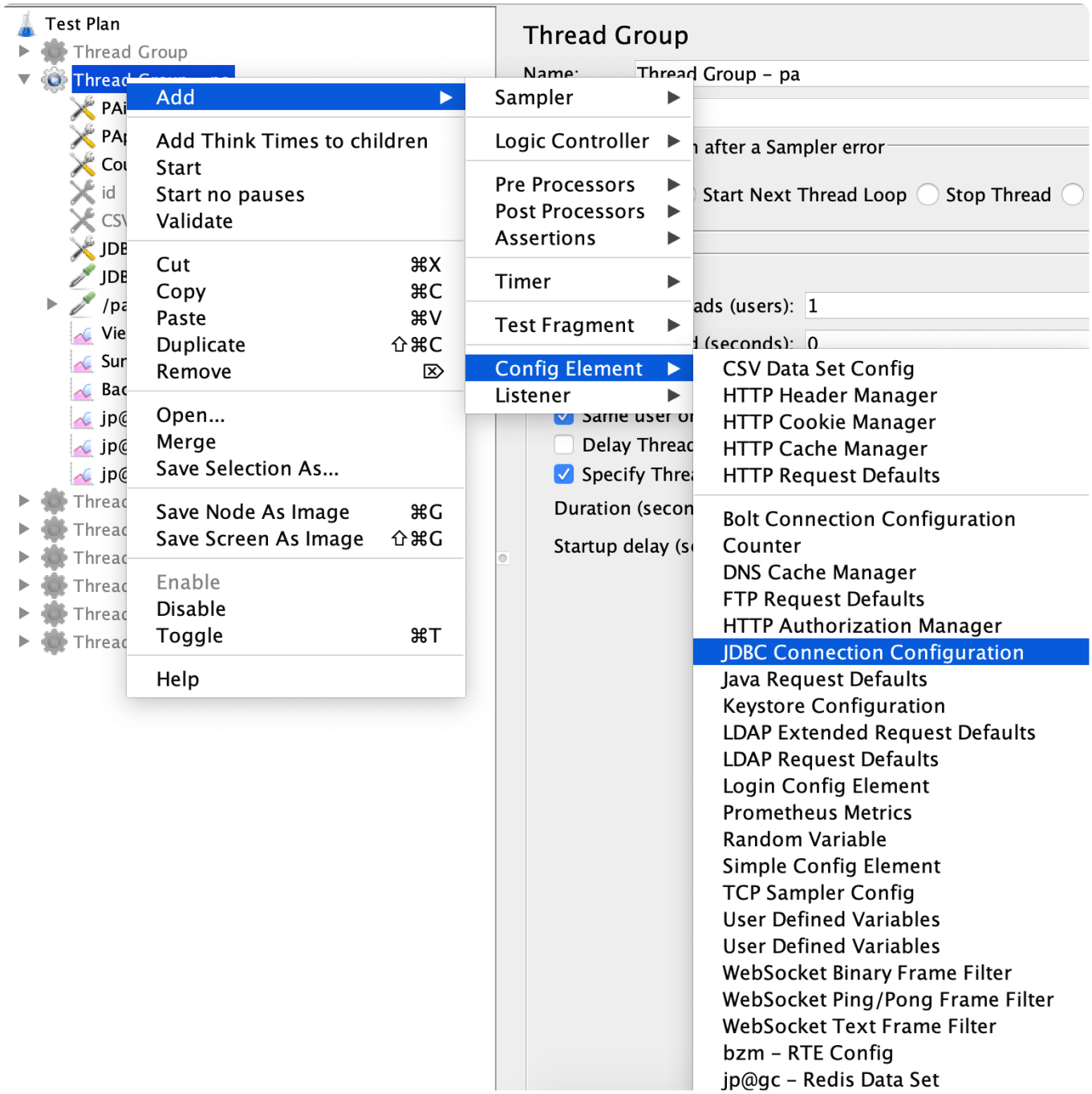
方法二：使用 Redis Data Set 组件。



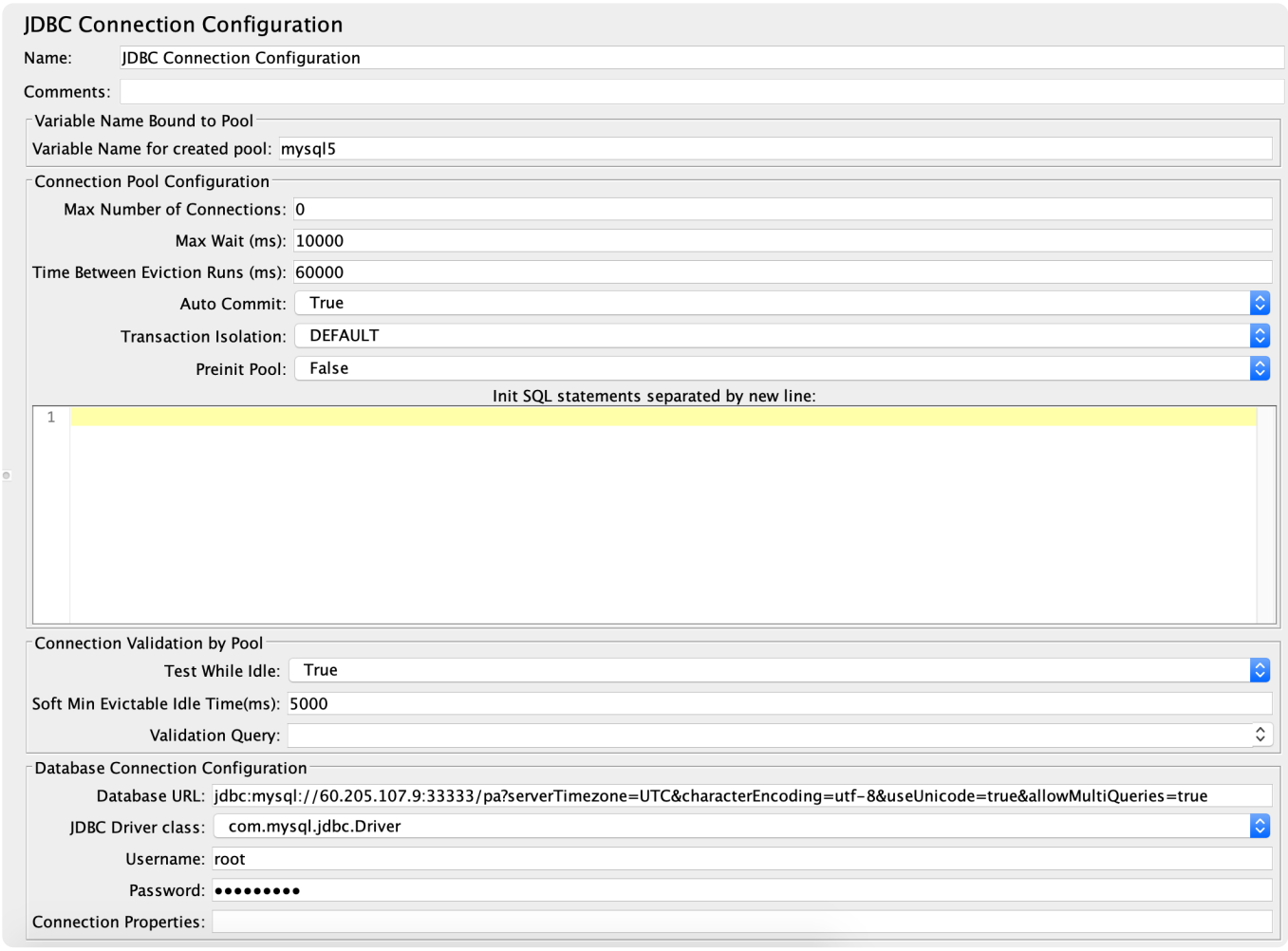
这两种方式都可以用 Redis 做参数化的数据源。

连接 MySQL 做参数化

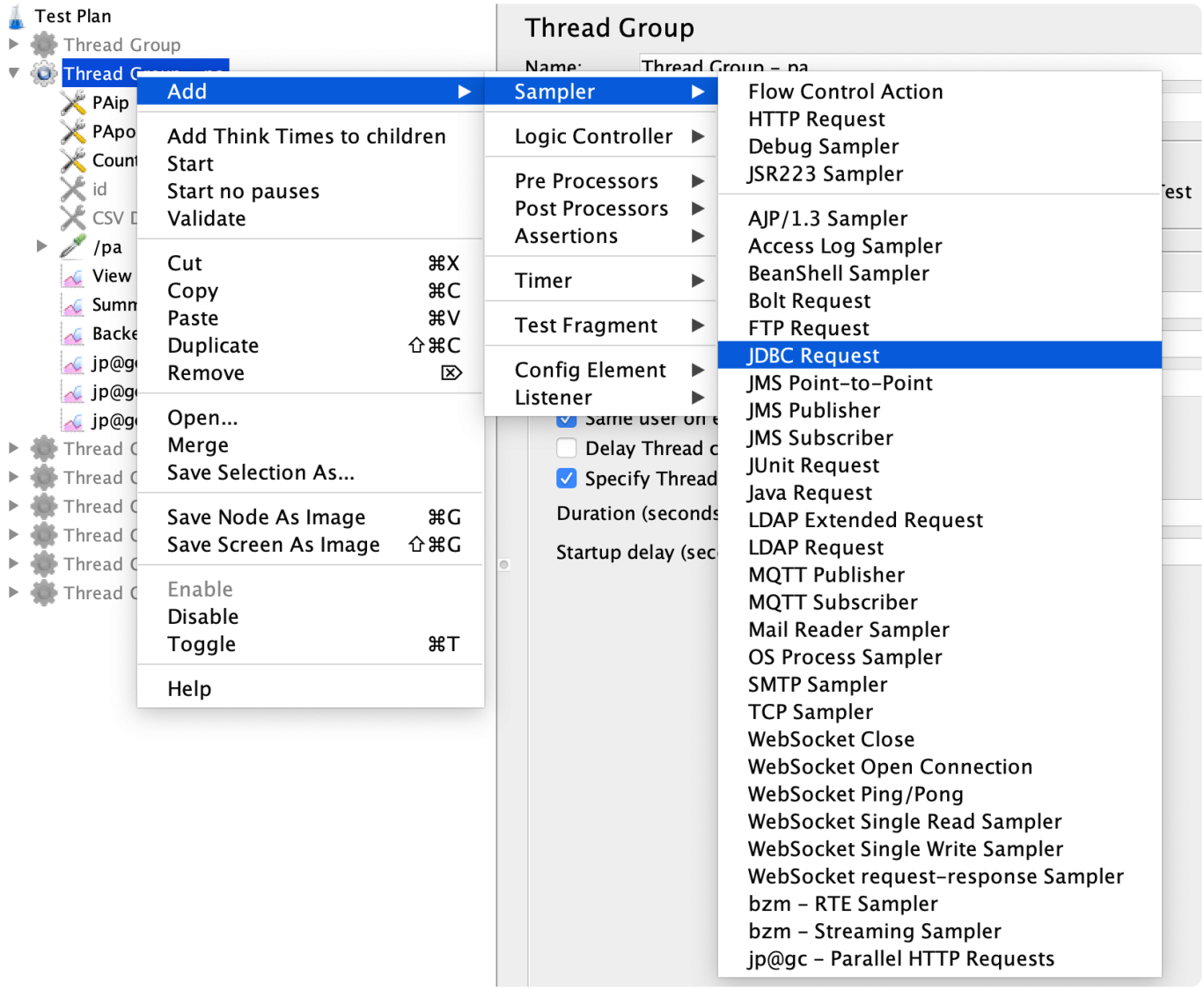
第一步：创建一个 JDBC Connection Configuration。



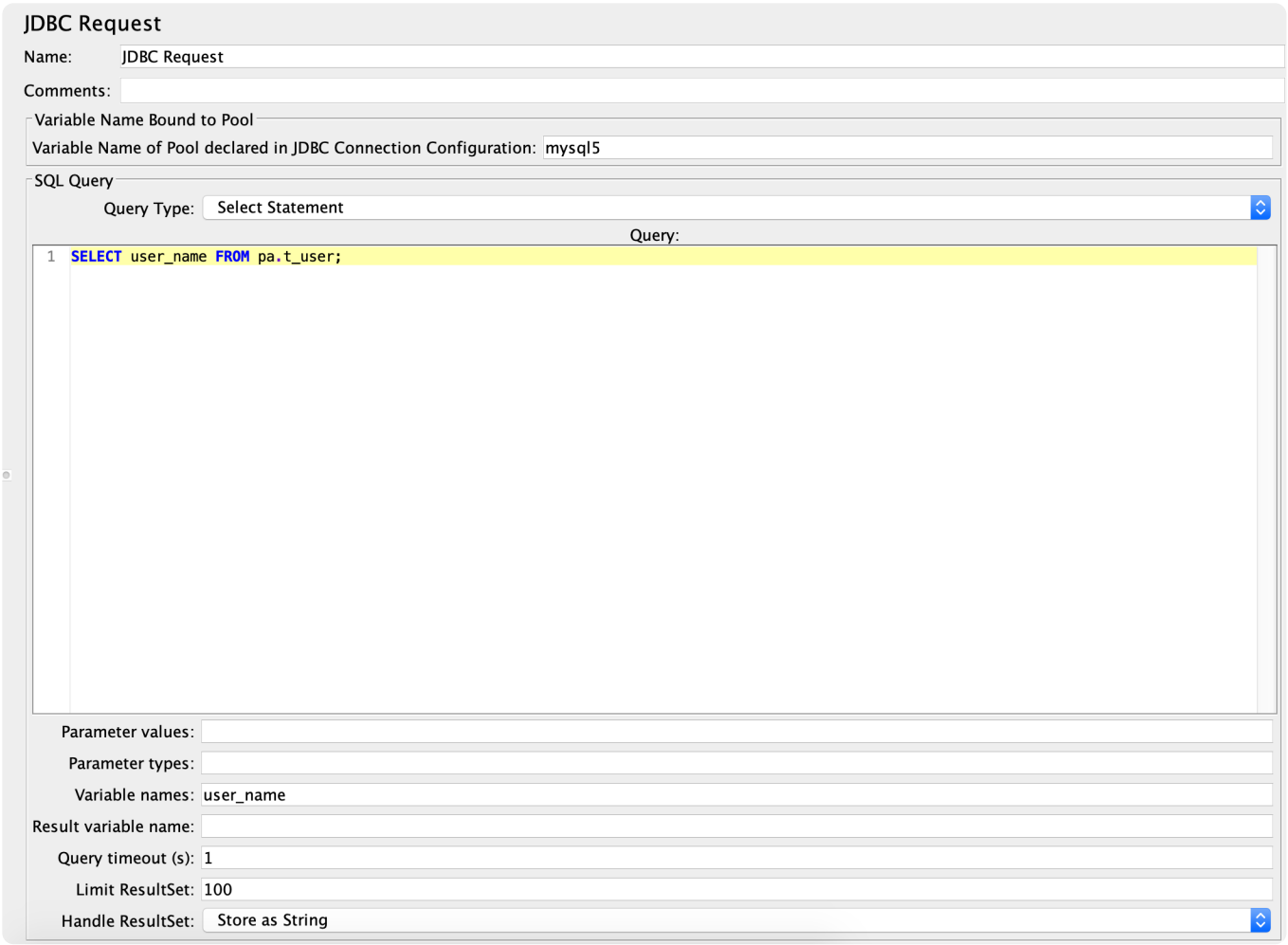
同时，配置好连接信息，比如用户名密码等：



第二步：创建一个 JDBC Request。



用 JDBC Request 把数据取回来：



第三步：用 ${user_name}引用参数。

完成上述三步，我们就实现了用数据库的方式做参数化。

知道了 RESAR 性能工程中需要什么样的数据后，我们接下来聊一下如何造数据。

如何造数据？

因为我们这个项目中的电商平台是开源的，数据库也完全是空的，系统中没有任何的数据。所以，我们虽然只是实现了电商的主流程，但需要的数据量仍然不少。这些数据包括：

用户数据；

地址数据；

商品数据；

订单数据。

下面我们具体考虑一下数据量应该怎么设置。

根据我们第 5 讲中的性能方案，登录 TPS 如果是每秒 150，并且如果按容量场景的需求，在场景连续递增时，大概在 20 分钟内（这是一个经验值，在具体的场景执行中也会有变化）会递增到最大值，然后再执行 10 分钟，也就是说总时间大概为 30 分钟。

但是因为场景是递增的，一开始我们并没有要求达到 150TPS，同时登录场景 TPS 最大值能达到多少，我们现在也没法预知。根据经验来看的话，登录的 TPS 在当前的硬件环境中，就算是不走缓存，达到三、四百应该是没有多大问题的。

如果按最大 400TPS 来算，跑半个小时，需要的数据量就是 54 万，而我们造出来的用户量要远远大于这个量级。这里我们就先造 200 万的用户量，因为地址的数据量肯定大于用户的数据量，所以会多于 200 万。

我们先查一下当前的数据库中有多少数据量，再确定要造多少数据。

数据类型	数据量
用户数据	10
地址数据	3
商品数据	29
订单数据	47

这个数据量级明显是不够的，太少了。下面我们来看看怎么造出那么多的数据量。

我们造的数据主要分为两种：用户数据和订单数据。

登陆用户

对于登陆用户数据而言，我们要先了解表结构，因为造出来的数据只有符合业务逻辑才能使用。我们先看一下用户表结构和数据。

用户表：

Info Columns Indexes Triggers Foreign Keys Partitions Grants DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra	Comments
id	bigint(20)		NO			select,insert,update,references	auto_increment	
member_level_id	bigint(20)		YES			select,insert,update,references		
username	varchar(64)		YES	utf8	utf8_general_ci	select,insert,update,references		用户名
password	varchar(64)		YES	utf8	utf8_general_ci	select,insert,update,references		密码
nickname	varchar(64)		YES	utf8	utf8_general_ci	select,insert,update,references		昵称
phone	varchar(64)		YES	utf8	utf8_general_ci	select,insert,update,references		手机号码
status	int(1)		YES			select,insert,update,references		帐号启用状态:0->禁用；1->启
create_time	datetime		YES			select,insert,update,references		注册时间
icon	varchar(500)		YES	utf8	utf8_general_ci	select,insert,update,references		头像
gender	int(1)		YES			select,insert,update,references		性别：0->未知；1->男；2->女
birthday	date		YES			select,insert,update,references		生日
city	varchar(64)		YES	utf8	utf8_general_ci	select,insert,update,references		所做城市
job	varchar(100)		YES	utf8	utf8_general_ci	select,insert,update,references		职业
personalized_sign...	varchar(200)		YES	utf8	utf8_general_ci	select,insert,update,references		个性签名
source_type	int(1)		YES			select,insert,update,references		用户来源
integration	int(11)		YES			select,insert,update,references		积分
growth	int(11)		YES			select,insert,update,references		成长值
luckey_count	int(11)		YES			select,insert,update,references		剩余抽奖次数
history_integration	int(11)		YES			select,insert,update,references		历史积分数量

id	member_level_id	username	password	nickname	phone	status	create_time	icon	gender	birthday	city	job	personalized_signat...
19	4	7d4test00	$2a$10$X/1iNLMJ5pC2nRS1DpWPweFEDlER...	性能实...	15252897981	1	2020-12-04 11:13:15	https...	1	2020-12-04	北京0	test	test00146
20	4	7d5test00	$2a$10$YrVWaXw7HKE8AdNIDf2XZedkz4ElrG...	性能实...	15198017398	1	2020-12-04 11:13:15	https...	1	2020-12-04	北京0	test	test00141
21	4	7d6test00	$2a$10$aHmoxW6YkB6MjjfCRqC6l.7yblezNum...	性能实...	13936815329	1	2020-12-04 11:13:16	https...	1	2020-12-04	北京0	test	test00121
22	4	7d7test00	$2a$10$rb9jpFLWH9RgvvRbChx0BOkzv0.2ew...	性能实...	15624324112	1	2020-12-04 11:13:17	https...	1	2020-12-04	北京0	test	test00118
23	4	7d8test00	$2a$10$6uFaZH4sURWFKusmI5E3eOgUaJpLj...	性能实...	18796907350	1	2020-12-04 11:13:18	https...	0	2020-12-04	北京0	test	test00115
24	4	7d9test00	$2a$10$XGAqKwKI/2kihk.Yp4WYRuceXwNAwt...	性能实...	19951332865	1	2020-12-04 11:13:23	https...	1	2020-12-04	北京0	test	test00145
25	4	7d1test0...	$2a$10$WfKodwaain8jNn9GbPsHDO5hf7.u5R...	性能实...	15069840156	1	2020-12-04 11:17:12	https...	1	2020-12-04	北京0	test	test00128
26	4	7d2test0...	$2a$10$41T5AuvRqg4/P.jpbe6qi.l.edli7ISt9Nw...	性能实...	16693173787	1	2020-12-04 11:17:16	https...	0	2020-12-04	北京0	test	test00196
27	4	7d3test0...	$2a$10$lTYfJL301rrWRFAATnflvOOqlnV8iN4W...	性能实...	18623241328	1	2020-12-04 11:17:17	https...	1	2020-12-04	北京0	test	test00140
28	4	7d4test0...	$2a$10$wb5zHygqymY.Dz1xTWb8qujDy8k.Liw...	性能实...	15689864339	1	2020-12-04 11:17:17	https...	0	2020-12-04	北京0	test	test00190
29	4	7d5test0...	$2a$10$k0bUrequ/etYW7nJ5YmMl.WmLczPFz...	性能实...	16697330391	1	2020-12-04 11:17:17	https...	1	2020-12-04	北京0	test	test00178
30	4	7d6test0...	$2a$10$QzHNBFkyW7J9COPFcUdoy.xiVtc4G6...	性能实...	13092483196	1	2020-12-04 11:17:17	https...	1	2020-12-04	北京0	test	test00121
31	4	7d7test0...	$2a$10$Slym0Rd8fLVznrWGDmpaJODml7Q.8...	性能实...	13682334826	1	2020-12-04 11:17:18	https...	0	2020-12-04	北京0	test	test00143
32	4	7d8test0...	$2a$10$JSpsfqwhon60PPzndMIFUurnM8HIoM...	性能实...	13700735922	1	2020-12-04 11:17:18	https...	0	2020-12-04	北京0	test	test00193
33	4	7d9test0...	$2a$10$//nZRz1A57X916spxoPgl.5hv4nzgu6Y...	性能实...	17860730376	1	2020-12-04 11:17:18	https...	0	2020-12-04	北京0	test	test00182
34	4	7d1test0...	$2a$10$BH9TT1wJqOGNMzlkZ38II.JKbbCg51...	性能实...	18051427353	1	2020-12-04 11:18:42	https...	1	2020-12-04	北京0	test	test00131
35	4	7d2test0...	$2a$10$adw4aTjv9zf9sM2dZLgJpuo7cTHV5q4...	性能实...	18074480484	1	2020-12-04 11:18:45	https...	1	2020-12-04	北京0	test	test00157
36	4	7d3test0...	$2a$10$KXbeRWB6W4hXAkZWwuiNluxfcHlaK...	性能实...	13792961700	1	2020-12-04 11:19:00	https...	1	2020-12-04	北京0	test	test00112
37	4	7d4test0...	$2a$10$DxL.6ATm0qa3GTH.0Rg83uTfqH01IQ...	性能实...	13589764923	1	2020-12-04 11:19:14	https...	1	2020-12-04	北京0	test	test00186
38	4	7d1test0...	$2a$10$QCGY3BnfNFRSxQqUMG3OiOWRGN...	性能实...	14762909985	1	2020-12-04 11:20:00	https...	0	2020-12-04	北京0	test	test00143
39	4	7d2test0...	$2a$10$rN9Nn7TBJP5Cc7uoZYp0MOaStN7Lo...	性能实...	18264893540	1	2020-12-04 11:20:03	https...	1	2020-12-04	北京0	test	test00192
40	4	7d3test0...	$2a$10$GVJaYr0Va6IzCueyae9XEuB.Zi1pTL6...	性能实...	18483077865	1	2020-12-04 11:20:04	https...	1	2020-12-04	北京0	test	test00158

地址表：

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra	Comments
id	bigint(20)		NO			select,insert,update,references	auto_increment	
member_id	bigint(20)		YES			select,insert,update,references		
name	varchar(100)		YES	utf8	utf8_general_ci	select,insert,update,references		收货人名称
phone_number	varchar(64)		YES	utf8	utf8_general_ci	select,insert,update,references		
default_status	int(1)		YES			select,insert,update,references		是否为默认
post_code	varchar(100)		YES	utf8	utf8_general_ci	select,insert,update,references		邮政编码
province	varchar(100)		YES	utf8	utf8_general_ci	select,insert,update,references		省份/直辖市
city	varchar(100)		YES	utf8	utf8_general_ci	select,insert,update,references		城市
region	varchar(100)		YES	utf8	utf8_general_ci	select,insert,update,references		区
detail_address	varchar(128)		YES	utf8	utf8_general_ci	select,insert,update,references		详细地址(街道)

id	member_id	name	phone_number	default_status	post_code	province	city	region	detail_address
25	12	hjn4j	18001860481	0	97011	北京	7dGruop性能实战	7dGruop性能实战区	fezm6吉地12号,1607866085592
26	12	leuwf	14971504805	0	60911	北京	7dGruop性能实战	7dGruop性能实战区	dlflr吉地12号,1607866085592
27	12	k3i6z	18384124539	0	39233	北京	7dGruop性能实战	7dGruop性能实战区	0qbgx吉地12号,1607866085592
28	12	g8o8g	19896357285	0	82834	北京	7dGruop性能实战	7dGruop性能实战区	v8ndo吉地12号,1607866085592
29	13	qxt1b	15740660892	0	59896	北京	7dGruop性能实战	7dGruop性能实战区	nreot吉地12号,1607866085592
30	13	w9iji	15814130316	0	51993	北京	7dGruop性能实战	7dGruop性能实战区	8slii吉地12号,1607866085592
31	13	on6tp	18840803813	0	15327	北京	7dGruop性能实战	7dGruop性能实战区	as7cm吉地12号,1607866085592
32	13	2m1vx	18422062171	0	24261	北京	7dGruop性能实战	7dGruop性能实战区	hac1f吉地12号,1607866085593
33	14	xsbmx	16609805104	0	97318	北京	7dGruop性能实战	7dGruop性能实战区	jtkc0吉地12号,1607866085592
34	14	yyw0d	14522338542	0	39783	北京	7dGruop性能实战	7dGruop性能实战区	f8hb0吉地12号,1607866085592
35	14	dzf1k	15329516342	0	88890	北京	7dGruop性能实战	7dGruop性能实战区	x0du3吉地12号,1607866085593
36	14	vxsir	14565905762	0	46675	北京	7dGruop性能实战	7dGruop性能实战区	g4fnh吉地12号,1607866085593
37	15	0pz05	13955754425	0	65673	北京	7dGruop性能实战	7dGruop性能实战区	mec6h吉地12号,1607866085593
38	15	nd785	15049660704	0	13926	北京	7dGruop性能实战	7dGruop性能实战区	nyht5吉地12号,1607866085593
39	15	vwasb	17762674379	0	04420	北京	7dGruop性能实战	7dGruop性能实战区	7bose吉地12号,1607866085593
40	15	6ngm1	13168959407	0	17810	北京	7dGruop性能实战	7dGruop性能实战区	kcfm9吉地12号,1607866085593
41	16	nlk7t	15026394842	0	21866	北京	7dGruop性能实战	7dGruop性能实战区	auv6e吉地12号,1607866085593
42	16	e054d	15265246733	0	98875	北京	7dGruop性能实战	7dGruop性能实战区	c0m8r吉地12号,1607866085593
43	16	ndi5k	14567822295	0	42481	北京	7dGruop性能实战	7dGruop性能实战区	5djo4吉地12号,1607866085593
44	16	kbxt0	13671164695	0	77740	北京	7dGruop性能实战	7dGruop性能实战区	dvoka吉地12号,1607866085593
45	17	blo9n	13709490219	0	49093	北京	7dGruop性能实战	7dGruop性能实战区	8hjtc吉地12号,1607866085593
46	17	vi73b	17893024939	0	16513	北京	7dGruop性能实战	7dGruop性能实战区	2vbf2吉地12号,1607866085593

根据我的经验，**造数据时不要往数据库里直接写存储过程插数据，除非你非常清楚表之间的关系，并且存储过程又写得非常溜。**否则你会把数据库弄得一团乱，最后不得不在数据库的表里改数据，这是非常被动的做法。在这里，我推荐你使用接口直接调用来造数据，这个操作比较简单，也比较安全。

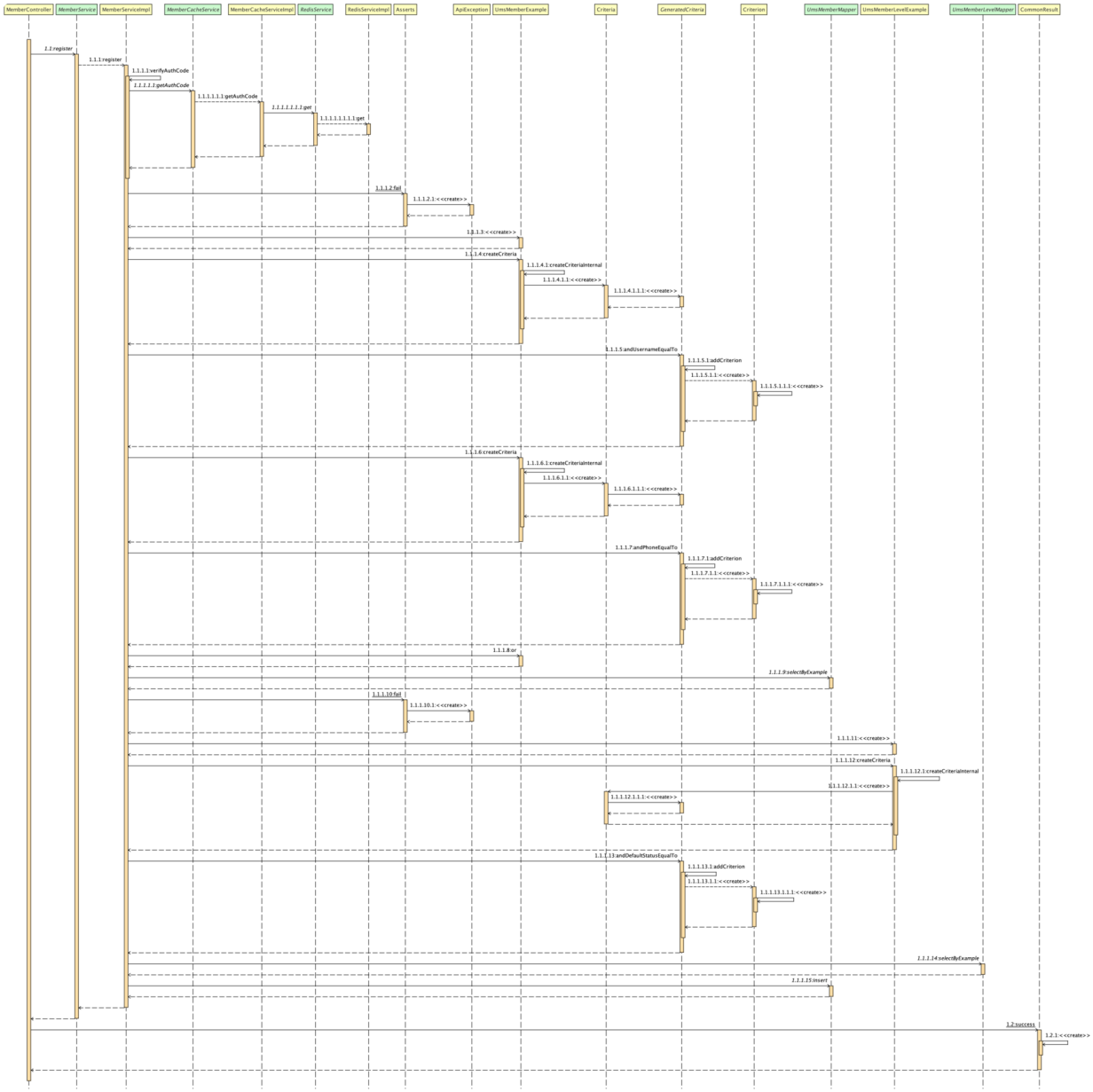
如果你想用代码来造数据，那就需要做下面这些分析。

在这里，我们的用户表和地址表之间是有对应关系的，你可以通过下面这段代码查看到，地址表中的 MemberID 就是用户 ID 号。

复制代码

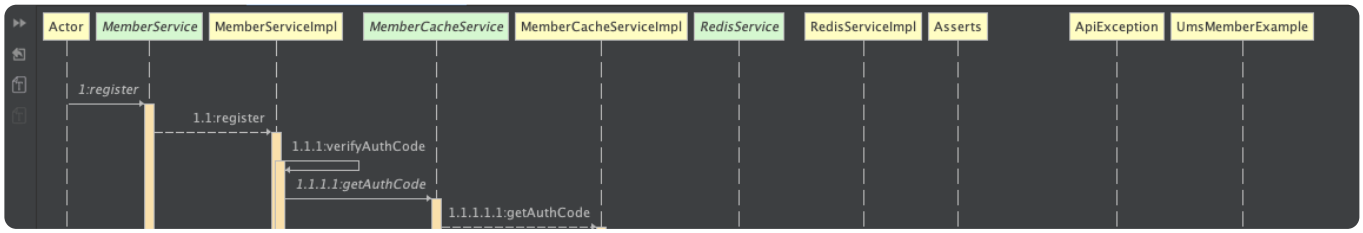
```
@Override
public int add(UmsMemberReceiveAddress address) {
    UmsMember currentMember = memberService.getCurrentMember();
    address.setMemberId(currentMember.getId());
    return addressMapper.insert(address);
}
```

其实造用户数据就是实现注册流程。你可以先分析下用户注册的代码，直接把其中的注册代码部分拿过来用就行了。具体调用代码如下：



看到这里，你可能会想，造数据需要关心注册流程吗？其实如果我们是调接口造数据，就不需要；但如果写代码开启了多线程来造数据，我们就需要了解接口之间的调用关系了。

下面我们截出中间一部分来分析它们的调用关系：



因为注册用户表中的密码都是加密的，所以我们可以通过注册用户实现类代码，如下：

复制代码

```
1 @Override
```

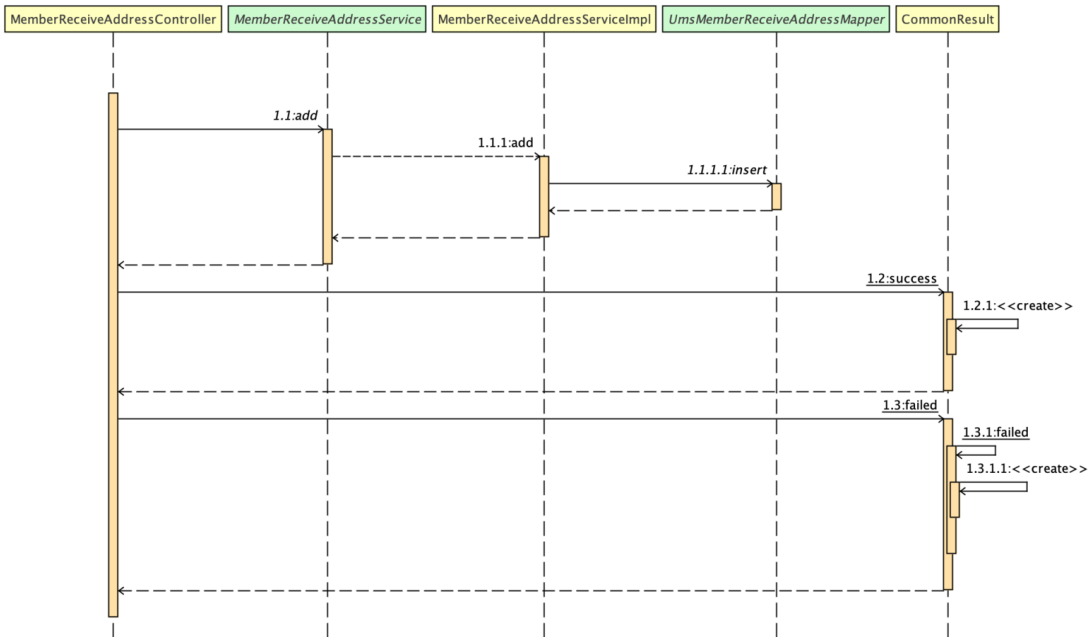
```
public void register(String username, String password, String telephone, Strin
.............................
    //获取默认会员等级并设置
    UmsMemberLevelExample levelExample = new UmsMemberLevelExample();
    levelExample.createCriteria().andDefaultStatusEqualTo(1)  ;
    List<UmsMemberLevel> memberLevelList = memberLevelMapper.selectByExample(l
    if (!CollectionUtils.isEmpty(memberLevelList)) {
        umsMember.setMemberLevelId(memberLevelList.get(0).getId());
    }
    //插入用户
    memberMapper.insert(umsMember);
    umsMember.setPassword(null);
}
```

了解了上面的内容之后，我们就可以直接写一段代码来造用户数据了，具体请见：《造用户代码.java》

有了用户数据，我们还需要下单用户的地址等详细信息，只有这样才能完成下单。所以，接下来我们就开始分析怎么造出可以下单的地址数据。

用户地址

首先，我们要根据用户地址资源路径找到 Controller 层，查看用户地址的代码调用关系，如下：



然后找到用户地址的关键代码：

复制代码

```
@Override
public int add(UmsMemberReceiveAddress address) {
    UmsMember currentMember = memberService.getCurrentMember();
    address.setMemberId(currentMember.getId());
    //插入地址
    return addressMapper.insert(address);
}
```

从这段代码中，我们可以观察到这几个信息：

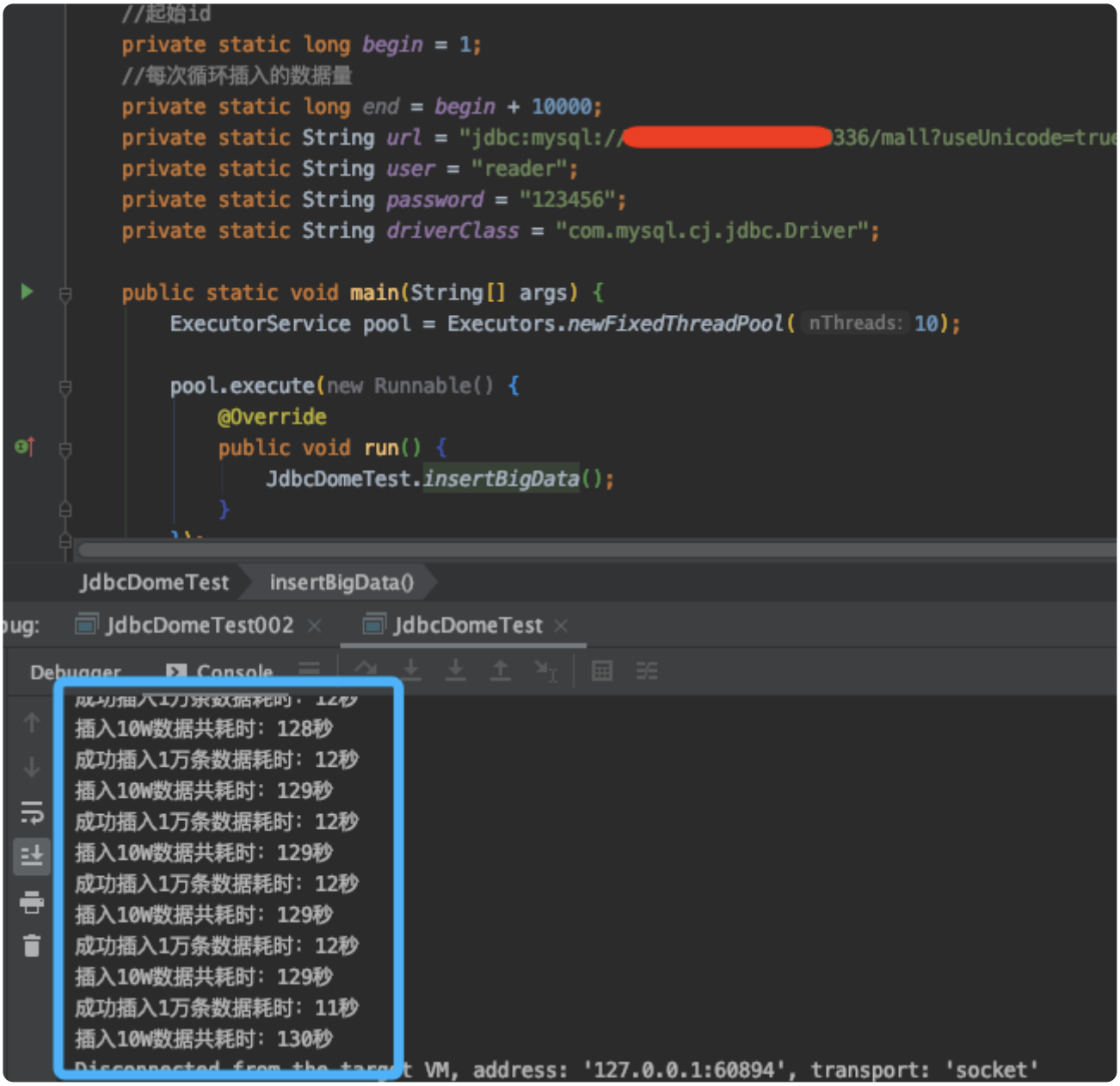
调用地址接口需要用户登陆态，通过登陆态来解析用户 ID 号；

用户 ID 号是地址代码中的 MemberID 号；

用户 ID 号是自增加。

具体参考请见《造用户地址代码.java》。

通过上面的代码编写，然后再开启 Java 线程池与多线程，我们就可以把基础数据快速造完了。下面是造用户地址数据的时间记录（每台电脑配置不一样，用的数据也许会有差异）：



通过以上手段，我们最后造出如下数据量：

数据类型	数据量
用户数据	2433398
地址数据	2503347
商品数据	119364
订单数据	170

表中的订单数据会在做基准场景时补充上去。等这些数据量都有了，我们在容量场景中就有了足够的铺底数据。

总结

在这节课里，我们一起学习了性能场景中的数据到底应该做成什么样子。对于造数据而言，方法有很多，我们不用拘泥于某种造数据的手段，只要能快速造出足够的数据量就好。在 RESAR 性能工程中，性能场景需要两类数据：铺底数据和参数化数据。其中，铺底数据需要满足这三个条件：

一定要造出符合生产量级的数据量；

数据量要真实模拟出生产的数据分布；

数据要真实可用。

参数化数据需要满足这两个条件：

参数化数据量要足够；

要符合真实用户的输入数据。

有了以上这些知识，我们就不会在造数据时出现混乱的情况了。

课后作业

这就是今天的全部内容，最后给你留两个思考题吧：

1. 为什么要造出符合生产量级的数据量？
2. 为什么参数化时要用符合真实用户的输入数据？

记得在留言区和我讨论、交流你的想法，每一次思考都会让你更进一步。

如果这节课让你有所收获，也欢迎你分享给你的朋友，共同学习进步。我们下一讲再见！

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 06 | 如何抽取出符合真实业务场景的业务模型？

下一篇 08 | 并发、在线和TPS到底是什么关系？

精选留言 (6)

写留言

wfw123
2021-04-06

老师，通常我们执行性能脚本的时候不会执行一次，如果按照上面30分钟100tps的场景，对于不可重复使用的数据，如果我们每次执行前做一次数据，太不易用啦，但是不造数据还不行，那么我们如何来解决这样的数据问题呢？

展开

作者回复: 做数据库备份回滚哇。

1 1

garlic
2021-04-14

造出生产量级数，确保应用，数据库，缓存在生产压力下运行；参数化用户输入数据，保障了数据多样性，造数很容易造成数据同化严重，热点集中，不能再现真实，老师400TPS跑半个小时 是否应该是72万数据量？

展开

作者回复: 如果数据在每个用户使用时是唯一的，那就应该是72万的数据量。

Geek_bbe4c8
2021-04-12

“对于唯一性数据（比如用户数据）来说，我们需要使用多少参数化数据是非常容易计算

的。比如一个运行半小时的场景，TPS 如果是 100 的话，那就需要 18 万的数据量”
老师，你这段话的意思是，要模拟每一个压测请求都是不同的用户发起的，这里需要构造18万个不同用户？

展开

作者回复: 是的。

Geek_bbe4c8
2021-04-12

老师，帮忙看下我的理解有没错。
铺底数据就是已存在的数据
参数化数据就是压力测试发起请求时产生的新数据

展开

作者回复: 参数化数据有两种。一种是用户输入的但数据在铺底数据中没有；还有一种是铺底数据中必须有的数据，比如说用户名之类的。

一只胖虎
2021-04-09

老师真实环境下单都是要真正的支付钱的，这个怎么解决

展开

作者回复: 如果对方提供测试系统就可以连。如果不提供，那只能mock了。

道长
2021-04-08

老师，我想问下你说尽量模拟真实用户效果，但像我们测试web一般都是忽略了css,js等直接测试主接口，这种是否合理呢？

展开

作者回复: 对后端的压力来说，是合理的。
如果从用户的角度来说，显然是不合理的。 所以我们要做相应的换算。 请看第8篇中的换算。